Interactive Procedural Modelling of Coherent Waterfall Scenes

Arnaud Emilien, Pierre Poulin, Marie-Paule Cani, Ulysse Vimont

Volume xx (200y), Number z, pp. 1–15

Interactive Procedural Modeling of Coherent Waterfall Scenes

A. Emilien[1,2†] P. Poulin[1‡] M.-P. Cani[2§] U. Vimont[2¶]

[1] LIGUM, Dept. I.R.O., Université de Montréal [2] LJK, Université Grenoble-Alpes, CNRS & Inria

Figure 1: *An example of a waterfall scene, the Iron Hole, on the island of La Réunion. Left: Photo of the real site © Serge Gélabert. Center: Our result after 10 minutes of interactive procedural modeling, starting from a similar terrain model. The scene contains 36 elements interconnected by pools and rivers, deforming the terrain and controlling the flow. Right: Visualization of the control elements that we used to create the waterfall network.*

Abstract

Combining procedural generation and user control is a fundamental challenge for the interactive design of natural scenery. This is particularly true for modeling complex waterfall scenes where, in addition to taking charge of geometric details, an ideal tool should also provide a user with the freedom to shape the running streams and falls, while automatically maintaining physical plausibility in terms of flow network, embedding into the terrain, and visual aspects of the waterfalls. We present the first solution for the interactive procedural design of coherent waterfall scenes. Our system combines vectorial editing, where the user assembles elements to create a waterfall network over an existing terrain, with a procedural model that parameterizes these elements from hydraulic exchanges; enforces consistency between the terrain and the flow; and generates detailed geometry, animated textures, and shaders for the waterfalls and their surroundings. The tool is interactive, yielding visual feedback after each edit.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1. Introduction

Procedural modeling is a great paradigm for automatic modeling of complex objects and scenes, and it has been applied to many problems including terrains, trees, buildings, streets, and cities. In addition to its efficiency for generating complex details, its power lies in its ability to ensure that certain

† arnaud.emilien@inria.fr
‡ poulin@iro.umontreal.ca
§ marie-paule.cani@inria.fr
¶ ulysse.vimont@inria.fr

constraints are respected (for instance from a physical, biological, or architectural viewpoint), making it much easier for the user to create plausible models.

However, when one has a very specific goal in mind, controlling the many intricate parameters of these automatic procedures can be very cumbersome. In such situations, it would be great to have access to an interactive system enabling the user to handle the coarse design, while the system automatically ensures consistency and support for the procedural generation of all the details. In this paper, we apply this paradigm of interconnected procedural generation and interactive user-control, to the difficult task of modeling coherent waterfall scenes.

Waterfalls offer some of the most beautiful settings in nature. Despite this, no easy-to-use method for designing waterfall scenes has been developed so far in computer graphics. One solution consists of using physically based simulation of fluids on top of a modified terrain. Unfortunately, modeling a terrain in order to produce specific waterfalls is an extremely daunting task. Fluid simulation depends on slopes, collisions, riverbeds, source influx, water properties, flow speed, etc. All the work of deforming a terrain does not ensure the final appearance of a waterfall, as these mutually dependent constraints and the governing physical laws are complex and highly nonlinear. Another potential solution consists of manually creating a few waterfalls with standard modeling tools, using manifold meshes and/or particles, and then positioning them along the terrain and connecting them by streams. In this long and tedious process, the artist also needs to manually maintain the consistency between the terrain and waterfalls, as well as the self-consistency of the waterfall network.

In this paper, we combine interactive and procedural methods to enable fast and easy design of plausible waterfall scenes. Our solution, based on a new interactive procedural model for flowing-water networks, allows users to easily shape complex waterfall scenes while automatically enforcing the physical consistency of the results, both in terms of hydraulic flow and of plausible embedding into the terrain. Coherence is enforced at several levels, ensuring that the flow always goes downstream, the user-edited water network follows the terrain (or the terrain is adapted to the network), the flow is distributed in the network without loss or gain of volume, and that the type of waterfall and the appearance of water is adapted to the flow. Our main contributions include:

- a slope-flow diagram-based classification of waterfalls;
- three parametric models for designing waterfall elements;
- a procedural method for ensuring waterfall network consistency;
- automatic methods for locally adapting user-input water trajectories to the terrain and/or the terrain to the flow.

All these contributions combine into a framework allowing an artistic approach to river and waterfall design. The resulting scene could either be used as a synthetic environment for games or films, or as an initial setup for further refinement through physically based fluid simulation when more sophisticated simulation effects are needed.

2. Previous Work

Many types of procedural methods [STBB14] have been designed for specific types of objects, including terrains [GGG*13], plants [LRBP12], buildings [LWW08], cities [CEW*08], road networks [GPGB11], and villages [EBP*12]. To our knowledge, no previous work addresses the procedural modeling of waterfall scenes. We therefore focus our literature review on the modeling of rivers, waterfalls, and terrains, which are key elements for our problem.

2.1. Rivers and Waterfalls

Most previous methods for creating running-water elements, from rivers to waterfalls, consist in simulating shallow water flows over an existing terrain [KM90, HW04, KW06, TMFSG07, YNBH09, LH10, BSW10, CM10]. In particular, particle systems are heavily used for generating waterfalls. However, even when optimized with hierarchical or screen-space methods [BSW10], these approaches suffer from the inherent complexity of simulating networks of running water over large environments. Moreover, they only provide indirect control on the nature of waterfalls and on the trajectories of streams, the latter being dictated by the geometry of the underlying terrain.

To increase user control over complex waterfalls, Sakaguchi et al. [SDZ*07] introduce pre-tuned particle systems designed to capture specific types of waterfalls. These systems are assembled by the user to generate visually complex scenes. Another solution [GCZ*06] directly creates waterfall scenes by placing polygonal primitives over the terrain, and renders them using animated textures. However, in both cases, the consistency within the flow network and between the terrain and waterfall elements needs to be manually ensured by the user.

As with these two last approaches, we move away from pure simulation methods in order to give more control to the user. However, we do so while maintaining the main benefits of physically based simulation, namely automatically ensuring a coherent result. This includes both that flow strengths are coherent between the elements of the waterfall network and that each flow segment is geometrically consistent with respect to the underlying terrain. The way our user-designed vector elements control flow trajectories is similar in spirit to the sketch-based approach from Bhat et al. [BSHK04] for editing videos of 2D flows, although their method is based on video processing only and does not address the problem of plausibly embedding water flows into a 3D terrain.

Although we do not use their simulation methods, we rely on rendering techniques introduced in previous work to display the aspects of waterfall scenes in real time: Yu et al. [YNBH09] derive an animated texture from the motion of particles simulated at the surface of a river, which effectively provides the appearance of a complex fluid. These textures can be augmented with their flow skirting around stones and borders [YNS11]. Instead, Van Hoesel [vH11] tiles flow textures and modulates their application on the water surface polygons according to the flow speed. His method proves to be quite efficient, compact, and effective. Zhu et al. [ZIH*11] introduce an interactive flow and diffusion editor with a sketching interface. The flow textures we use combine ideas from all of these approaches, and are enhanced by simple forms of particle systems, inspired by work from Holmberg and Wünsche [HW04].

2.2. Terrains

Allowing users to design a specific waterfall scene over some input terrain requires automatic ways of adapting the terrain in order to ensure consistency with the flow. We therefore briefly review previous methods for terrain generation and editing.

Most terrain models are represented using a 2.5D heightfield that is well adapted to GPU processing, enabling the handling of very large environments [LH04, BN08]. In contrast, Peytavie et al. [PGGM09] introduce a specific piling model for capturing complex terrain structures, such as overhangs and caves that can be observed at the top of free-falls or near pools at their bottom. In our work, we instead adapt a horizontal displacement method [GM01] to capture overhangs over a standard heightfield representation.

Many solutions exist for terrain generation and editing [SDKT*09], from fully procedural methods to those combining examples or textures with sketch-based interaction. We focus on the latter, since they could be adapted to control some local deformations of the terrain.

Hydraulic erosion methods [BTHB06, MDH07, ŠBBK08] simulate the effect of water flows on the progressive sculpting of terrains, which results in very realistic landscapes. However, here the control is indirect since the flow depends on the current terrain. Therefore, this approach is not suitable for our problem. Closer to our concerns, Génevaux et al. [GGG*13] present a procedural method for generating terrains based on hydrology: a realistic terrain is fully generated from dense hydraulic graphs. Unfortunately, their approach does not directly address our goal: we are instead looking for plausible ways to locally edit an existing terrain in order to make it consistent with user-designed flow networks.

Re-using existing terrains to match user specifications, such as sketches, is generally accomplished using texture-based approaches, where heightfield patches from the input terrain are combined to match user constraints [ZSTR07, TGM13]. We are rather interested in local, feature-based editing of the input terrain, to have it match the range of slopes and local geometry consistently with user-designed flow networks. We are therefore inspired by feature-based terrain generation methods [HGA*10, BMV*11, GMS09], and we adapt them for the first time to the editing of an existing heightfield.

3. Overview of our Method

The key goal of our method is to leave coarse design of waterfall scenes in the hands of the user, while providing automatic ways to generate plausible and detailed results.

In the real world, flow networks formed by complex waterfalls, such as the one in Figure 1(left), include free-falls, segments where running water remains in contact with the terrain, as well as pools. Moreover, each of these segments can be of many different types, from rivers to rapids for the ones in contact with the terrain, and from plunges to cataracts for free-falls. Having to manually select plausible types for each segment of a full network would be both tedious and require specialized knowledge from the user.

Below we propose a two-level classification for segments of waterfall networks, enabling us to leave the choice of low-level classes to the user, while automatically computing the most appropriate running-water types from quantitative information, such as the slope of the underlying terrain and the intensity of the flow. The processing pipeline that we use for modeling a waterfall scene, based on this analysis, is presented next.

3.1. Two-level Classification for Running Water

Waterfall scenes are comprised of three types of elements: running-water segments that remain in contact with the terrain, free-fall segments where water is in the air, and pools that receive water from the free-falls. Our goal is to provide some coarse, intuitive control to the user, and we leave the choice of these three classes, *contact*, *free-fall*, and *pool*, to the user during interactive design.

In contrast, we would like to free the user from the manual and explicit determination of the precise characteristics that each running-water segment should take, because this can be determined in a more plausible way using an automatic procedure. After studying the existing classifications of streams and falls, we designed a new, slope-flow classification that matches our goals, as explained next.

Running water can take on many forms, from rivers to rapids, and from plunges to cataracts. See Figure 2 for an illustration of these classes. Several classifications of waterfalls are proposed by hydrologists, geologists, and artists, in order to capture this variety:

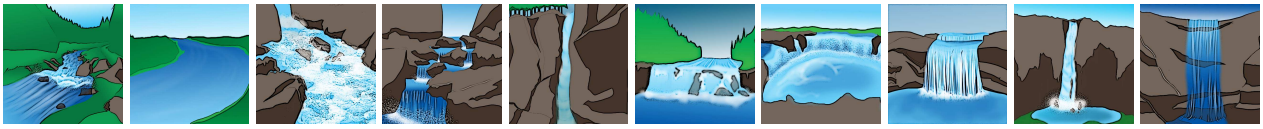

Figure 2: *Artistic drawings illustrating the different types of running water and free-falls that can be found in nature.*

- volume-based classifications of waterfalls [Bei06] sort waterfalls into classes by using a logarithmic scale over the volume of water in the air at a given time. Although easy to compute from quantitative information, this classification provides little clue on the visual aspect of the fall. Moreover, it is restricted to free-falls, and therefore does not fully meet our needs;
- geometric classifications, such as the one found in the Waterfall Lover's Guides [DD06, Plu05] or the one depicted in Figure 2, analyze the different types of geometries that can be observed in nature. However, they only provide visual information. No quantitative measurement is proposed to automatically compute the class that a running-water segment belongs to.

In this work, we would like to classify waterfall segments from quantitative information, while getting visual clues enabling us to generate plausible 3D representations for each segment. We therefore decided to augment the geometric classification of Figure 2 with quantitative evaluation of the classes, as in volume-based approaches. However, we need measures applicable to both running water in contact with the terrain and to free-falls, and therefore our solution is different.

By studying the existing geometric classifications and looking at many real cases, we noticed that the geometric type of running water mostly depends on two important quantitative parameters: the flow value (defined as the volume of water per second traveling through a cross-section of the segment), and the local slope of the terrain. Intuitively, when the flow decreases, a river becomes a stream, a cascade becomes a horsetail, and a free-fall cataract becomes a ledge. Meanwhile, if a given water flow is running on terrains of increasing slope, a river tends to become a rapid, and then eventually a block.

To provide a quantitative classification, we define the different classes of running-water segments as regions in a unified slope-flow diagram. This is done as follows: we first used real examples to find a consistent set of seed values for typical elements of each of the visual types depicted in Figure 2. These representative slope and flow values are listed in the left columns of Table 1. The regions associated with each class are then defined using a qualitative Voronoi segmentation in slope-flow space. The resulting classification is depicted in Figure 3. Note that the types of water flows on the left of the division in red belong to contact waterfall segments, while those on the right belong to free-fall waterfall segments.

We also used the visual classifications serving as reference to associate visual parameters with each type of elements in our classes, given in the columns to the right of Table 1. These values will be used for generating the appropriate visual representation for each element of the waterfall scene.

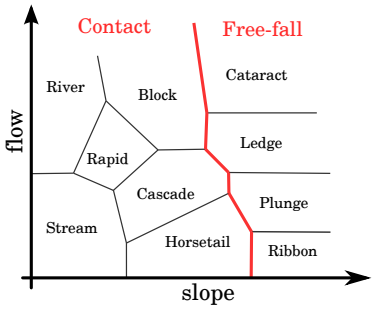

Figure 3: *Our classification of waterfall types.*

3.2. Processing Pipeline

In the remainder of this paper, we define a *waterfall network* as a network of *waterfall segments* and of *pools*. *Waterfall segments* can either be of type *contact* (in contact with the terrain) or of type *free-fall* (in the air).

Our processing pipeline, based on the two-level classification we just defined, develops as follows (see Figure 4):

1. The user starts by creating a waterfall scene, building an *oriented vectorial controller network* $\mathcal{U}$ over an existing terrain. This is done using three vectorial controllers (*contact*, *free-fall*, and *pool*)all based on Cardinal splines. *Contact* and *pool* are interactively created by using control points while free-falls are parabolas, automatically parameterized by their start and end points. During interaction, controllers are not constrained to be placed in contact with the terrain (since the terrain will be adapted later according to the user design), but the flow is constrained to go downhill along each segment of the controller network. While the default flow intensities on each segment of the network can be interactively tuned by the user, a

Type	Slope	Flow	Foam	Rocks	Dist.	Part.
Ribbon	$\pi/2$	1	0	0.3	0	0
Plunge	$\pi/2$	2	1	0.4	0	0.2
Ledge	$\pi/2$	5	0	0.5	0	0.5
Cataract	$\pi/2$	8	1	1	1	1
Stream	$\pi/16$	2	0	0.2	0	0
River	$\pi/32$	5	0	0.2	0.1	0
Rapid	$\pi/16$	5	0.2	0.5	0.5	0
Cascade	$\pi/8$	3	0.5	1	1	0
Horsetail	$\pi/4$	2	1	0	0	0
Block	$\pi/4$	6	1	0	0.2	0

Table 1: *Classification of running-water segments that may appear in a waterfall network. The coordinates (slope, flow) give the position of the class seed in the slope-flow diagram of Figure 3. Slope is an average inclination in radians, while flow is expressed in "flow units", which gives an informal notion of relative proportions between waterfall types. Foam, rocks, disturbance, and particle are parameters ($\in [0,1]$) used in our procedural generation of geometry and for rendering.*

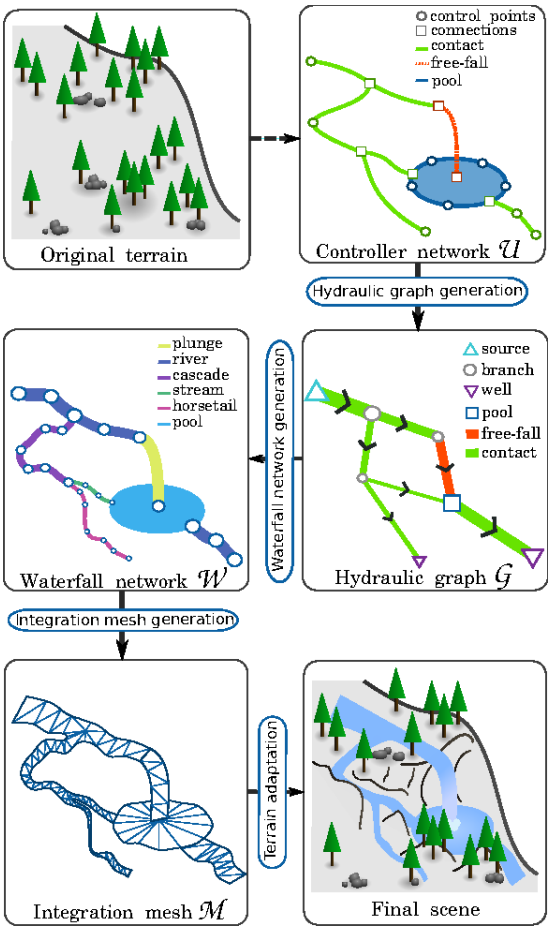

Figure 4: *Processing pipeline used for creating waterfall scenes. The first step is the creation by the artist of the controller network $\mathcal{U}$. Then, a hydraulic graph $\mathcal{G}$ is generated; the width of an arc encodes flow quantity. The waterfall network $\mathcal{W}$ is then generated with a subdivision algorithm, and the waterfall types are determined. Finally, the integration mesh $\mathcal{M}$ is generated and used to deform the terrain and generate the procedural details.*

consistent hydraulic graph $\mathcal{G}$ (with fully consistent flow values) is automatically computed at the end of the interactive modeling process. (See Section 4)

2. The next step is the generation of the *waterfall network* $\mathcal{W}$, which uses the coarse water trajectories from the controller network and the flow information from the hydraulic graph to define a more precise representation of the waterfall. In addition to directly using the control curves that they defined, the user has the option of further refining the geometry of the network through an automatic procedure that locally adapts running-water trajectories to the underlying terrain. Each curve of the network is then divided into a number of water segments, whose sub-class is determined by slope-flow classification from Section 3.1. Finally, the last geometric parameters, such as flow width and depth, are computed for each segment of the waterfall network. (See Section 5)
3. Lastly, the 3D representation for the waterfall network, called the *integration mesh*, is generated and embedded into the scene through appropriate local deformations of the terrain. Although they can include large changes, such as digging a canyon to allow a stream to find its way downhill (in the extreme case where the user designed water segments go through a mountain), constraints applied to the terrain are mainly aimed at automatically adding all the details that make the scene plausible: this includes borders along streams, riverbeds, and the creation of overhangs behind free-falls. Appropriate decorative elements such as trees and rocks are generated at this stage, using the distance to the closest riverbed and the class of the waterfall segment it belongs to as design guidelines. Rendering attributes are also set from the class of each segment. (See Section 6)

Note that we chose to compute a coherent hydraulic graph $\mathcal{G}$ (Step 1 of the pipeline above) based on the controller network $\mathcal{U}$, i.e., from the coarse trajectories defined by the user, before refining these trajectories into a waterfall network $\mathcal{W}$. This enables us to use consistent flow values in Step 2, while refining water trajectories. This helps us, for instance, to prevent large rivers from being refined into a series of short twists when adapted to the terrain, although a smaller stream would be allowed to be more winding. The fact that our algorithm interleaves geometric com-

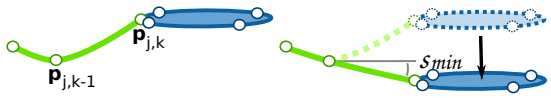

Figure 5: *Minimum slope constraints are imposed on each curve of the controller network, to make it consistent with the user-defined flow direction. Left: A controller network. Right: The algorithm verifies that the minimum slope s_{min} is respected between the contact control points $\mathbf{p}_{j,k-1}$ and $\mathbf{p}_{j,k}$. Since this is not the case, $\mathbf{p}_{j,k}$ is lowered. All the subsequent controller points will also respect this constraint. In this figure, the pool control points are therefore lowered.*

putation with consistency checks leads to more plausible results. The remainder of this paper details the three aforementioned steps of the pipeline.

4. From User Input to a Coherent Hydraulic Graph

We will discuss how the user creates the controller network $\mathcal{U}$ (Section 4.1), how we generate the associated hydraulic graph $\mathcal{G}$, and how we compute its flow (Section 4.2).

4.1. Controller Network Creation

The user builds a controller network $\mathcal{U}$ (Figure 4) by creating and manipulating different vectorial elements $\mathcal{V}_i$ and by interconnecting them.

We define *controllers* using a Cardinal spline with a controller type $\alpha_i \in \{$ *free-fall*, *contact*, *pool* $\}$. The splines are composed of a series of control points $\Delta_i = \{\mathbf{p}_{i,k}\}$, which can be connected to control points from other controllers to create a controller network (Figure 4 Step 1). Depending on α_i, controllers are set differently: a controller *contact* is created by positioning a series of control points $\mathbf{p}$ over the terrain. This controller is used to create all the elements that remain in contact with the ground, such as rivers. The *pool* controller has been introduced to create flat polygons, and is used to create pool contours. Because a controller *pool* is flat, the user first positions a horizontal plane in space, and then traces a contour over the plane using control points. Finally, the *free-fall* controller is used to create an element that loses contact with the ground. Consequently, only a start point and an end point can be positionned on the terrain, with the lower point being typically placed inside a *pool*. The user associates a flow direction to *free-fall* and *contact* curves.

During editing, point and curve magnets are used to facilitate the interactive creation of the connections between elements, like in most classical vectorial editing tools. Moreover, the user can insert, delete, and move control points on each curve, and cut or merge controller curves.

Projection. To facilitate the adaptation of the controller network to the underlying terrain, users are provided with a projection tool, which can be used to project control points of the Cardinal splines onto the terrain. If desired, they can also define an offset from the terrain's local height for each control point. When the projection tool is used for pools, which are constrained to be flat, the pool level is automatically set to the average height of all projected contour points.

Minimum slope. For the scene to remain consistent, the system must ensure that all slopes set to the controller network segments allow the water to flow downstream in the user-defined direction. This is accomplished using automatic correction of the user-defined positions, during a traversal of the controller network (see Figure 5): starting with the flow *sources* of the network, and traversing it in a topologically sorted order, we check if each control point $\mathbf{p}_{j,k}$ of curve Δ_j of each controller maintains the minimum slope s_{min} with respect to its predecessor $\mathbf{p}_{j,k-1}$. If not, point $\mathbf{p}_{j,k}$ is lowered down to match the constraint, i.e., for each connection, we check that all the outgoing nodes are lower than the incoming nodes, and update their positions if needed. Pool contour points are lowered, if necessary, according to the full set of free-fall segments coming into the pool.

4.2. Hydraulic Graph Generation

The hydraulic graph $\mathcal{G}$ (Figure 4 Step 2) is an oriented graph, generated from the controller network $\mathcal{U}$, in which the flow originates from the *sources* and exits by the *wells*. It is composed by a set of nodes $\mathcal{N}_j = (\beta_j, \gamma_j)$ and a set of arcs $\mathcal{A}_k = ((\mathcal{N}_i, \mathcal{N}_j), \eta_k, \gamma_k)$, where $\beta_j \in \{$ *source*, *well*, *branch*, *pool* $\}$ is its type, γ_j is the flow going through the arc or the node, and $\eta_k \in \{$ *contact*, *free-fall* $\}$ is the arc type.

For each controller $\mathcal{V}_i$ we create an arc if its type α_i is *free-fall* or *contact*, and create a node if α_i is *pool*. Nodes for *sources* and *wells* are introduced at the extremities of the graph, *branches* are created at the intersections between controllers.

4.2.1. Flow Computation

It would be time-consuming and non-intuitive in cases of failure to perform a complex physical simulation on the flow propagation in order to deduce the entire flow properties (intensity, speed, etc.) everywhere along the network. We were inspired by the interactive system of Zhu et al. [ZIH*11], and therefore simplified our hydraulic model by considering it as a "pipe-like" graph. This enables us to deal only, at this stage, with flow exchanges at the nodes of the hydraulic graph. In the remainder of this section we detail how to compute these flow exchanges using simple and intuitive functions, while providing a coherent visual appearance.

Flow consistency. A hydraulic node should be at equilibrium, i.e., its incoming flow should equal its outgoing flow. Expressing this at each node of the hydraulic graph leads to a system of interconnected equations. There is generally

an infinite number of solutions to this system. Therefore, instead of using a global solver to find consistent flow values, we solve them in succession for each node of the graph, exploring it in dependency order from sources to wells. This enables us to take into account the user specifications for the relative strength of the input flows, and to generate a solution that best matches the coarse geometric trajectories in terms of branching angles at each node. Our method for doing so is explained next.

Separating incoming flow into outgoing branches. The flow of an outgoing arc at a branch node $\mathcal{N}_i$ should depend on the angles between the inflow and outflow arcs in order to capture the natural course of water. For instance, we expect that most of the flow should follow its own original direction.

Let $\mathcal{X}_j$ be an input arc of $\mathcal{N}_i$, and $\mathbf{u}_j$ its incoming direction (Figure 6). We distribute its flow γ_j to each output arc $\mathcal{Y}_k$ with a direction $\mathbf{v}_k$ according to a normalized weight:

$$w_{jk} = 1 + (\mathbf{u}_j \cdot \mathbf{v}_k) \qquad \bar{w}_{jk} = \frac{w_{jk}}{\sum_k w_{jk}}.$$

Thus, the outgoing flow γ_k for arc $\mathcal{Y}_k$ is computed as

$$\gamma_k = \sum_j \gamma_j \bar{w}_{jk}.$$

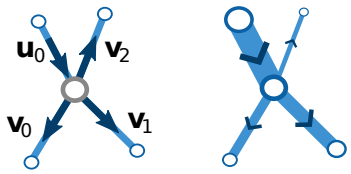

Figure 6: *Flow repartition in a branch. Left: Input and output directions. Right: Resulting flow exchange, drawn as segment thicknesses.*

Separating the flow out of pools. Because the shape of a pool could be complex, distributing the flow according only to the angles between inflows and outflows would not be realistic, while computing a full simulation would be computationally expensive and not suitable for an interactive system [ZIH*11]. Instead, we simply distribute inflows equally to all the outgoing arcs, except when we need to take the relative flow strength pre-set by the user into account for these branches.

5. Waterfall Network Generation

In this section, we discuss the generation of a waterfall network $\mathcal{W}$ from the controller network $\mathcal{U}$, which contains the waterfalls coarse trajectories, and the hydraulic graph $\mathcal{G}$, which contains the flow information. The waterfall network is composed of waterfall segments $\mathcal{S}_i$, interconnected by waterfall nodes $\mathcal{B}_i$.

We start the construction process by subdividing the controller trajectories and locally adapting them to the terrain (Section 5.1). Then, we extract from them the waterfall segments and nodes that form the waterfall network (Section 5.2). We compute the type of each segment using the classification from Section 3.1, and set its other parameters while taking flow and slope into account.

5.1. Subdivision and Adaptation to Terrain

In order to procedurally improve the curves created by the user, we provide a fractal-like subdivision scheme based on midpoint displacement, which takes the underlying terrain model into account (see Figure 7). Except for very intricate terrains, our subdivision scheme approximates the natural trajectory of a flow running down a slope while preserving the global shape of the user-defined curve.

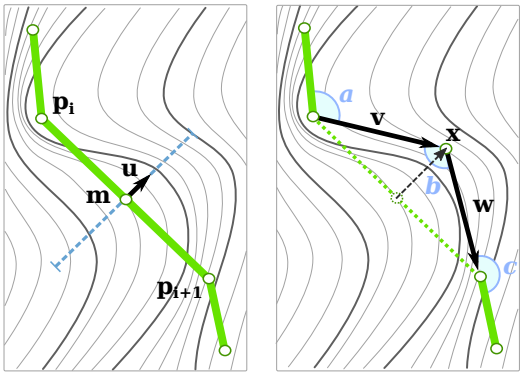

Figure 7: *One step of our recursive subdivision process for waterfall network segments: the segment formed by* $\mathbf{p}_i$ *and* $\mathbf{p}_{i+1}$ *is subdivided at* $\mathbf{m}$*, which is moved along perpendicular direction* $\mathbf{u}$*. The final position* $\mathbf{x}$ *is the point that minimizes our cost function* $C(\mathbf{x})$*.*

Let $\mathcal{A}_j$ be an arc of the graph $\mathcal{G}$, and $\mathcal{V}_i$ its corresponding controller, with Δ the points of the curve manually created by the user, and γ the flow going through the arc. Consider the segment formed by two consecutive points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. We subdivide this segment at its middle point $\mathbf{m}$ and move it along direction $\mathbf{u}$, perpendicular to this segment, horizontally. This creates two new segments, V and W, represented by normalized vectors $\mathbf{v}$ and $\mathbf{w}$. While the original midpoint displacement method applies a random displacement to $\mathbf{m}$ with a maximum amplitude τ, we instead search for $\mathbf{x} = \mathbf{m} + \lambda\mathbf{u}$, $\lambda \in [-\tau, \tau]$, minimizing the cost function:

$$C(\mathbf{x}) = w_g C_g(\mathbf{x}) + w_a C_a(\mathbf{x}) + w_r C_r(\mathbf{x})$$

where $C_g(\mathbf{x})$ is the gradient cost, $C_a(\mathbf{x})$ the angle cost, and $C_r(\mathbf{x})$ the random cost. The values w_g, w_a, and w_r are weight coefficients associated to the costs.

Gradient cost. The gradient cost favors paths that follow the slope of the terrain, and is defined as:

$$C_g(\mathbf{x}) = \frac{\int_V \|\mathbf{g}(\mathbf{t}))\| f_p(\mathbf{v},\mathbf{g}(\mathbf{t}))d\mathbf{t} + \int_W \|\mathbf{g}(\mathbf{t}))\| f_p(\mathbf{w},\mathbf{g}(\mathbf{t}))d\mathbf{t}}{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|},$$

$$f_p(\mathbf{v},\mathbf{g}(\mathbf{t})) = \left(-2\mathbf{v}\cdot\frac{\mathbf{g}(\mathbf{t})}{\|\mathbf{g}(\mathbf{t})\|}\right) + 1,$$

where vector $\mathbf{g}(\mathbf{t})$ is the gradient of the terrain elevation at position $\mathbf{t}$. By integrating the elevation gradient along segments V and W, and by using the scalar product between the gradient and the segment vectors ($\mathbf{v}$ or $\mathbf{w}$) as a penalty coefficient, the path will follow the slope and avoid obstacles (Figure 7). Moreover, instead of simply using the scalar product value as a penality coefficient, we use function f_p to penalize paths that follow the isolines (i.e., when the scalar product with the gradient is zero); climbing a slope is penalized even more severly. Consequently, this cost is low when $\mathbf{v}$ and $\mathbf{w}$ are aligned in the same direction as the slope of the terrain, and high otherwise.

Angle cost. The angle cost prevents undesirable sharp angles that could appear between two consecutive segments, because of their independent subdivisions. We take into account the angles a, b, and c (in radians), created at the introduction of new point $\mathbf{x}$ (see Figure 7), as:

$$C_a(\mathbf{x}) = \left(\frac{a}{\pi}\right)^2 + \left(\frac{b}{\pi}\right)^2 + \left(\frac{c}{\pi}\right)^2.$$

Random cost. Finally, to add fractal-like details on flat terrains, we use the random cost $C_r(\mathbf{x})$, which is negligible when the other costs are high.

The segments are recursively subdivided until their length is inferior to l. Displacement amplitude τ and detail size l are set as:

$$\tau = \|\mathbf{u}\|/2 \qquad l = \gamma/\sigma.$$

The minimum subdivision length l is proportional to the segment flow, where σ is a user parameter. This enables us to get a more detailed trajectory for small flow values, enabling streams to become more winding than rivers. In our prototype we use the following values: $\sigma = 1/2$, $w_g = 1$, $w_a = 0.1$, and $w_r = 0.2$ for all our examples.

5.2. Waterfall Network Construction

The waterfall network is composed of waterfall segments $\mathcal{S}_i = (\mathbf{u}_i, \gamma_i, \kappa_i, \delta_i, \varepsilon_i, \zeta_i)$ where $\mathbf{u}_i$ is the segment vector, γ_i is the flow going through the segment, $\kappa_i \in \{$ *stream*, *horsetail*, *cascade*, *rapid*, *block*, *river*, *ribbon*, *plunge*, *ledge*, *cataract* $\}$ is the waterfall type according to our classification (Figure 3), δ_i is the speed of the flow, and ε_i and ζ_i are respectively the width and depth of the riverbed. These segments are interconnected by waterfall nodes $\mathcal{B}_j = (\mu_j, \gamma_j, \zeta_j)$

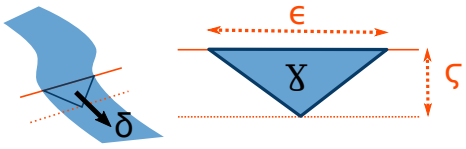

Figure 8: *Triangular cross section of a waterfall segment, with a constant flow.*

where $\mu_j \in \{$ *source*, *well*, *branch*, *pool* $\}$ is the node type, γ_j the total incoming flow, and ζ_j the depth.

Each segment vector $\mathbf{u}_i$ is directly extracted from the subdivided trajectories (Figure 4), and its flow γ_i is equal to the flow of its corresponding graph arc. All consecutive segments are connected by a branch node with only one input and one output. The other segments are connected by the waterfall nodes $\mathcal{B}_j$ constructed from the hydraulic graph nodes $\mathcal{N}_k$ and their associated controller $\mathcal{V}_l$. The remaining parts of this section describe how the other segment properties are computed.

5.2.1. Waterfall Segment Type

For each waterfall segment $\mathcal{S}_i$, we know its slope s_i and its flow γ_i. These values automatically determine the waterfall segment's type κ_i by casting its coordinates (s_i, γ_i) in the slope-flow graph of Figure 3. We use Voronoi cells, whose coordinates are detailed in Table 1, to determine to which type the coordinates belong.

Note that if the user did not use a plausible waterfall controller, e.g., if he created a free-fall on a flat terrain, or a contact on a very steep terrain, the waterfall segment may be outside of the valid range of values in the slope-flow graph. In this case, the parametric model is still set using the closest type, but the user is notified (i.e., the related segment is drawn in red). He can then either validate the current design (even if it is not fully realistic), or select more realistic controllers.

5.2.2. Waterfall Segment Properties

The final step in the generation of the waterfall network is to compute the last properties of each segment $\mathcal{S}_i$, i.e., its speed δ, its riverbed width ε and depth ζ, from the slope and flow information we have. We propose a resolution that leads to satisfying results while being intuitive and fast to compute. Our hypothesis is to consider the waterfall segment as a closed pipe, with a constant flow and a triangular cross section (Figure 8). In this case, the flow can be expressed by:

$$\gamma = \frac{\delta\,\varepsilon\,\zeta}{2}. \tag{1}$$

Since we have one equation with three unknowns, and complex inter-dependencies, a physically accurate solution

should rely on strong assumptions, which cannot be justified in our context. Instead, we decided to solve the system by providing simple and intuitive functions to compute these variables.

We first propose to solve for the speed as a simple linear function of slope s: $\delta = ks$, $k \in \mathbb{R}^+$. Then, we set the depth as a function f_d of the width and of the segment type as: $\zeta = f_d(\varepsilon, \gamma)$. Note that our slope constraint enforces that $s > 0$ everywhere for a waterfall segment. We used $k = 10$ in all our scenes. In our prototype, we use only one profile function, $f_d(\varepsilon, \gamma) = \frac{1}{2}\gamma$. Applying Equation (1), it is now possible to compute the width and to deduce the depth value.

While simple, our model efficiently provides plausible results using intuitive parameters. In Figure 9, the procedural component of our modeling system correctly handles a reduction of the input flow: when the upstream flow is manually reduced, the following free-fall flow reduces accordingly. The subsequent nodes in the graph are then affected. Note how the type of the outgoing element automatically changes, i.e., from ledge to plunge. Note that nothing prevents our method from being extended to account for more complex river profiles, e.g., as introduced by Génevaux et al. [GGG*13].

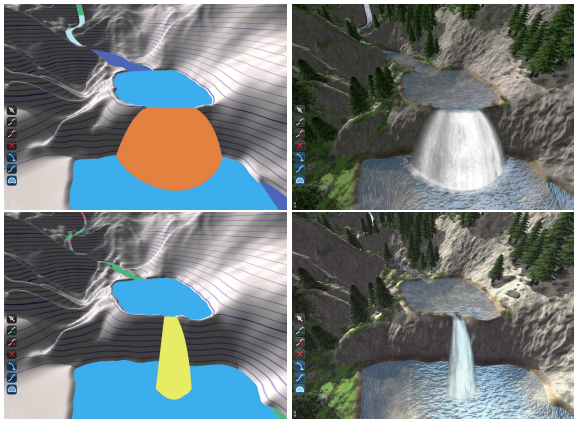

Figure 9: *Varying the flow within a graph. The waterfall elements are changed automatically, in conformity with the classification, and the visual aspects are immediately adapted to the changes.*

6. Terrains Adapting to Waterfalls

In this section, we discuss how to adapt the terrain to the waterfalls. First, we generate the waterfall integration mesh (Section 6.1), then we use it to generate vectorial constraints to deform the terrain (Section 6.2) and to generate additional maps (Section 6.3). The latter are used for the visual integration of the waterfall (e.g., texture changes on the terrain and on the waterfall) and for the generation of procedural decorations.

6.1. Integration Mesh Generation

The integration mesh is a 3D mesh defining the waterfall surface. It is composed of the surface meshes for all waterfall elements (Figure 10), which are generated using the waterfall network trajectories and the widths of the elements.

The meshes for the contact elements are computed by extruding the width ε along the segment trajectories. We carefully handle the meshing connections at branch intersections, so no meshes overlap one another. Free-fall meshes are extruded from the borders of their incoming segments (e.g., pool borders) and follow the free-fall element curve. Pool meshes are simply triangulated from their contours.

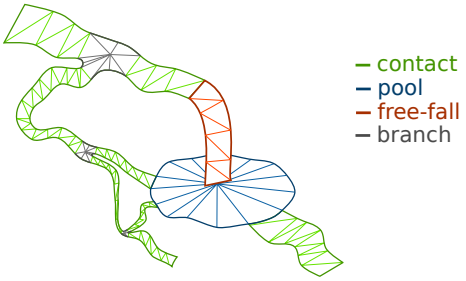

Figure 10: *Integration mesh composed by the individual waterfall element meshes.*

6.2. Constraint-based Terrain Deformation

The terrain is stored as a heightfield, on which a deformation map (displacement map) is applied to adapt the terrain to waterfalls. This deformation map is computed using a Poisson solution applied to vectorial constraints, as inspired by the terrain modeling method of Hnaidi et al. [HGA*10]. However, our solution differs from theirs in that it propagates deformations (i.e., height differences) instead of heights, and it generates three different types of constraints (border, riverbed, and overhang) using our procedural model. As a result of our approach, small details on the original terrain will remain on the terrain after deformation.

Border constraints. These constraints force the water to naturally follow waterfall trajectories, by defining height and gradient constraints on the mesh contour (Figure 11(top left)).

Let Ω be the contour of the integration mesh (Figure 12). For each point $\mathbf{p} = (x, y, z) \in \Omega$, we define d as the difference between the terrain height h and the height at the point, i.e., $d = z - h(x, y)$ (Figure 11(top center)). The difference d is the value that will be diffused within our solver of constraints. We also diffuse a gradient constraint to ensure that the neighborhood of the terrain just outside of the waterfall is higher than its border. Please refer to Hnaidi et al. [HGA*10] for more details about the diffusion algorithm.

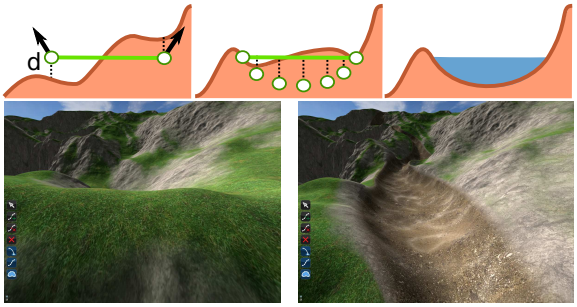

Figure 11: *Border and riverbed constraints. Top left: Border constraints. Center: Riverbed constraints. Right: Final terrain. Bottom left: Original terrain. Right: Deformed terrain with apparent riverbed.*

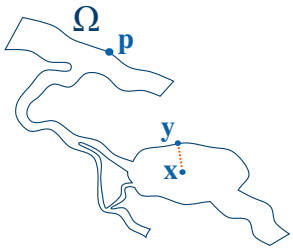

Figure 12: *Integration mesh borders.*

Riverbed constraints. The border constraints do not provide any control on the river profile, which should be a function of the river type (see Génevaux et al. [GGG*13]). For this reason, riverbed constraints are added.

For a given point $\mathbf{x}$ located inside the waterfall border (see Figure 12), we create a height constraint based on its distance to $\mathbf{y} \in \Omega$, the nearest point of the border, to which a profile function f_d is applied. The operation is repeated on a dense sampling basis. This defines a set of additional elevation constraints processed using our solver (see Figure 11(bottom)).

Overhang constraints. In order to create overhangs, we generate a horizontal displacement map based on the same diffusion technique. This map is then used to deform the terrain, as presented by Gamito and Musgrave [GM01].

Along the border at the top of a free-fall, we set a displacement constraint $\lambda\mathbf{u}$, where $\mathbf{u}$ is the free-fall direction and λ a constant defined by the user. In addition, we set a displacement constraint $-\lambda\mathbf{u}$ along the border of the receiving pool, under the free-fall.

As shown in Figure 13, these two constraints generate a flipped "S" curve, with the top of the overhang extending out of the terrain in the direction of the flow, and the bottom of the receiving pool extending into the cliff, due to erosion and falling rocks.

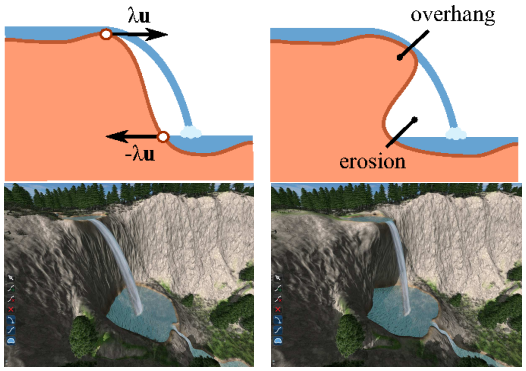

Figure 13: *Top: Horizontal constraints modeling overhangs. Bottom: Free-fall without and with an overhang.*

6.3. Procedural Decoration

To improve the integration of the waterfall into the terrain, we use several procedural decoration maps, generated using the footprint of the integration mesh and the waterfall elements type (Figure 16). A decoration map is computed by rendering the integration mesh viewed from above into a texture, similar to the road footprints of Bruneton and Neyret [BN08]. During this process, we render each submesh in a grey-scale map, depending on its type and on the map that is being computed.

Terrain decorations. Terrain decorations are generated using the water map, which corresponds to the integration mesh footprint. This map is used to mask the procedural seeding of trees and plants to prevent a generation within the water surface, and also to change the terrain texture to, for instance, a bedrock one.

Water decorations. Table 1 lists a set of parameter values depending on the waterfall type. By using these values as greyscale values during the map computations, we generate a foam map, a rock map, and a disturbance map. The foam map identifies the presence of foam on the water surface and is used to select the water diffuse texture; the rock map indicates the density of rocks to generate; and the disturbance map, the amplitude of the waves on the water surface (Figure 14). The variation of values depending on the waterfall type allows increases of the visual difference between them, and improves the appearance of the scene. Note that some filtering is applied to these maps to reduce visible transitions between different types.

Speed map. The speed map is a texture that represents the 2D speed of all water in contact with the terrain in the scene. It is used for animating the textures of the water surface.

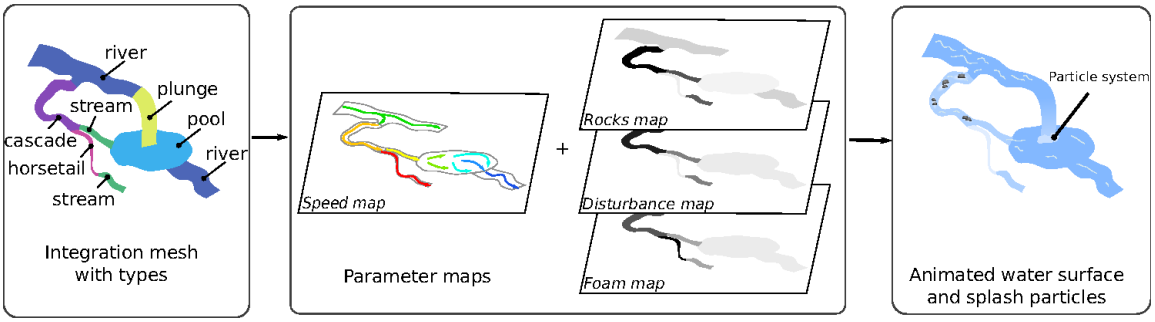

Figure 14: *Using the integration mesh and the waterfall types, we generate various maps used to render the waterfalls.*

Figure 15 shows how the speed of the water is computed depending on the type of the segment. We use three different approaches to compute the surface speed of *contacts*, *pools*, and *branches*. For a *contact*, the speed at point $\mathbf{x}$ is given by $\delta_{\mathbf{x}} = \delta_{\mathbf{c}} \left(1 - \frac{\|\mathbf{x}-\mathbf{c}\|}{\|\mathbf{b}-\mathbf{c}\|}\right)$ where $\mathbf{c}$ and $\mathbf{b}$ are the projections of $\mathbf{x}$ along the cross section on the main axis and on the shore respectively. For a *pool*, a fixed number of 2D fluid simulation steps [Sta99] are evaluated. For *branches*, we use 2D interpolation based on a standard technique of weighting by the inverse distance, where each point $\mathbf{p}_i$ is considered as a velocity constraint δ_i. The speed within a branch is given by $\delta_{\mathbf{x}} = \sum_i \omega_i \, \delta_i$. Interpolation weights ω_i are computed using the equation:

$$\omega_i = \frac{\tilde{\omega}_i}{\sum_i \tilde{\omega}_i} \qquad \tilde{\omega}_i = \prod_j \left(1 - \frac{\|\mathbf{x}-\mathbf{p}_i\|}{\|\mathbf{p}_j-\mathbf{p}_i\|}\right).$$

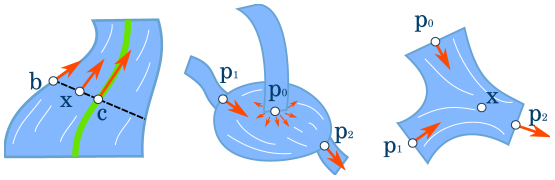

Figure 15: *Internal speed computation for a contact (left), a pool (center), and a branch (right).*

7. Implementation and Results

The system is implemented in C++, using OpenGL and GLSL Compute Shaders. The computations are performed on an NVidia 660GTX GPU and an Intel® Xeon® E5-1650 CPU, running at 3.20 GHz with 16 GB of memory. The system uses two threads: one CPU thread for the interface and computation control, and one CPU/GPU thread for the GPU computations and rendering.

Rendering. The waterfalls in our editor are rendered in real time using the integration mesh and the parameter maps computed earlier (Figure 14). We use the technique of "tiled directional flow" [vH11] for the animation of both the normal texture of the waves and the diffuse texture of the foam. The splashes at the bottom of the falls are rendered using particles emitted from the free-fall ends.

Figure 16: *Incremental representation of procedural decorations. In usual order: Terrain only, adding rocks, water, foam, speed map, final result with vegetation.*

Evaluation. Figures 17 and 18 show an overview of our system under editing, with different stages described in the caption of the figure. An accompanying video gives a much better understanding of our system in action, and illustrates several of the features described in the previous sections.

In Figure 1, we show a photo of a real waterfall network, and the result of a 10-minute session with our modeling system; we started with a terrain resembling the original real

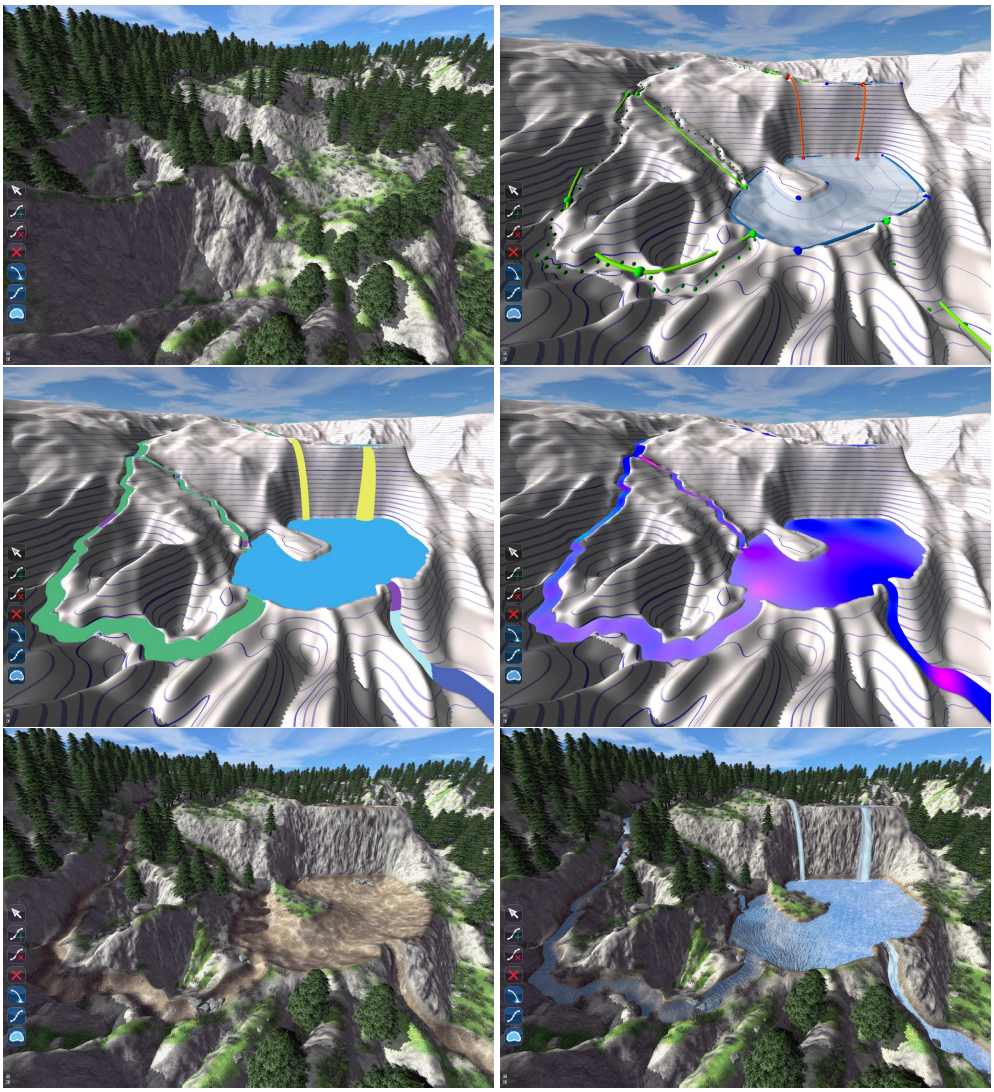

Figure 17: *Overview of the system. Top: Original terrain (left), creating the waterfall network (right). Middle: Generating the control mesh and deducing the types (left), resolving the speeds (right). Bottom: Adapted terrain (left), the final scene (right).*

terrain, but without riverbeds. The figure shows that coherent waterfalls similar to those on the photo can be easily modeled, while guaranteeing their physical plausibility.

We organized a user modeling session with two experienced digital artists. After a 20-minute training period, they were both able to create waterfall scenes such as the waterfall presented in Figure 19, all in under 30 minutes.

We asked to reproduce the scene of Figure 1 with classical modeling tools, such as Autodesk® Maya®, it took them more than two days to reach an equivalent level of detail for both the waterfall and terrain deformation; the longest part being the manual deformation of the heightfield to match the waterfall mesh. Of course, Maya® has not been designed to specifically model waterfalls, but this preliminary experiment shows how specialized tools can be beneficial. The artists were pleased by the ease of use of our system and its efficiency. However, they expressed a desire for finer control over the result.

Performance. Our system generates a complex waterfall network over a terrain in a few seconds (see Table 2). The number of elements does not have a huge impact on the computation time. Indeed, most of our algorithm uses a fixed-size grid, so its complexity is independent of the number of waterfall elements. When increasing the number of elements, only the time for mesh generation, the riverbed con-

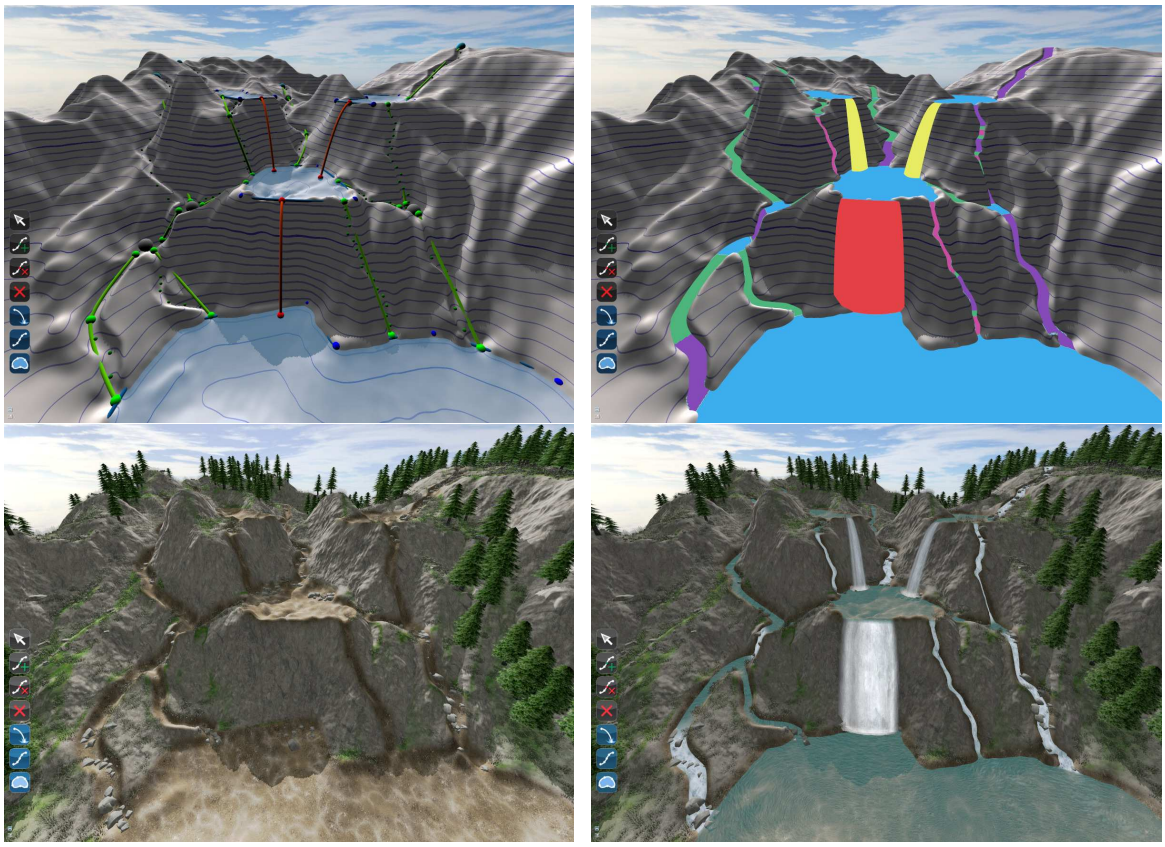

Figure 18: *Another example of waterfall scene. Top left: Controller network. Top right: Integration mesh with types. Bottom left: Deformed terrain. Bottom right: Final scene.*

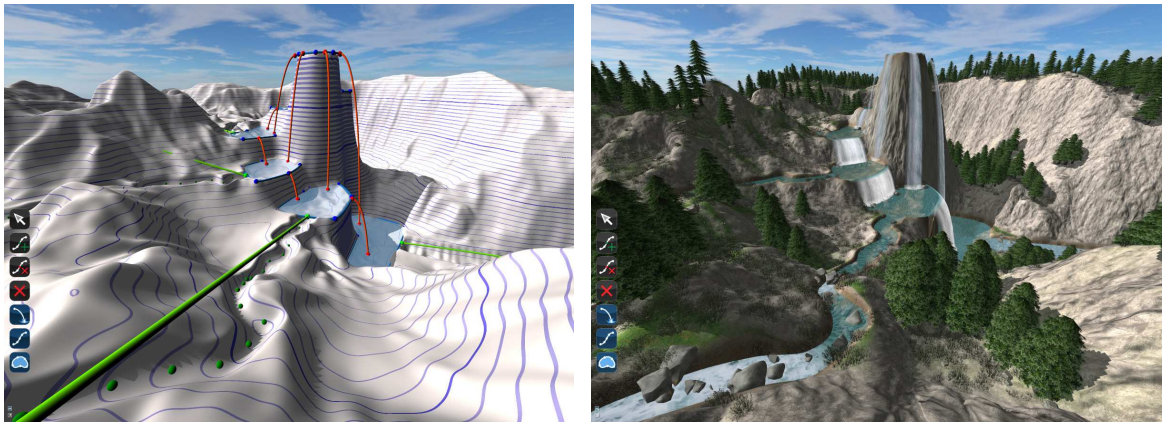

Figure 19: *Waterfalls modeled by one of our digital artists.*

straints dense sampling (sub-part of the terrain deformation algorithm), and the internal speed computation of *pool* vary noticeably.

In order to ensure the consistency of our results, each time an element is modified by the user, we reperform all computations. Many simple optimizations can readily detect what needs to be recomputed, and thus greatly improve the efficiency of our system; however, we felt that any such optimizations were not necessary in the current version of our prototype.

#		Computation time in ms						
Fig.	n	$\mathcal{G}$	$\mathcal{W}$	$\mathcal{M}$	Terrain	Speed	Maps	Proc.
1	36	2	2	112	758	258	428	284
17	17	1	1	177	677	384	404	297
18	29	1	1	77	871	264	494	324
19	26	1	2	30	1115	284	482	302

Table 2: *From left to right, the columns of the table list the figure number and the number of waterfall controllers, followed by computation times for the hydraulic graph generation ($\mathcal{G}$), waterfall network generation ($\mathcal{W}$), mesh generation ($\mathcal{M}$), terrain adaptation using a* 2048×2048 *resolution, speed map generation, details map (foam, disturbance, and rocks) generation, and procedural detail generation.*

Limitations. The first limitation of our method is that the terrain is only adapted locally, and therefore does not preserve any global hydraulic properties. Indeed, a waterfall network can be created at an unplausible location on the terrain, failing to respect natural river paths shaped by the terrain slope. While this may lead to unplausible terrains, it also gives more artistic freedom to the user, which we feel is an important property of our system.

The heightfield representation of the terrain is another limitation, as it prevents the creation of caves and underground waterfalls, although horizontal displacement maps [GM01] enable us to create overhangs. With support for stack-based terrains [PGGM09], our system could handle more complex terrain elements.

In addition, our adaptation method relies on a grid-based algorithm, which limits its application to relatively small terrains. In our examples, we used a 2048×2048 grid with a resolution of 10 pixels per meter. We could adapt our method to multi-scale editing and rendering [YNBH09], by dividing the terrain into several tiles, each calculated independently [ŠBBK08], but we have not yet done so.

Moreover, the recursive nature of our algorithm limits the adaptation of paths in complex cases. Consequently, a good position selection at a given step does not imply a good position for the final shape. For instance, if there are too many obstacles on the path, the heuristic will select a point to avoid them globally, but this selection may prevent further steps to avoid them. A solution inspired by algorithms for procedural roads [GPMG10], should be applicable to procedural river trajectories.

Finally, even if our algorithm supports interactive flow variations that change waterfall geometry and its adaptation to the terrain, we do not support a flow variation without changing the terrain adaptation. For example a drying-out waterfall or a river flood cannot be modeled at this moment with our system.

8. Conclusion

We presented the first interactive procedural modeling system for the design of waterfalls. Our system relies on a tight coupling of automatic generation and user interaction, where complex constraints and tedious tasks are handled by the procedural component of the system while enabling global and local user control. This leads to an improved user experience and the possibility for more creative modeling.

A number of aspects of our method are dedicated to the special case of waterfalls, including their categorization based on the slope-flow graph, and adapted tools for their interactive modeling. In fact, we are the first to present a system for the interactive design of coherent waterfalls. However, the methodology for the design of our system, with notably the way we interleave high-level user control with automatic processes to check consistency and add details, could be generalized to the modeling of many other natural sceneries, as well as to even less natural complex models.

Future work could focus on the preservation of the hydraulic properties of the terrain during its deformation, or tackle the development of volumetric algorithms allowing the creation of truly 3D waterfall networks.

Acknowledgements

We thank the anonymous reviewers for their feeback, helping substantially to improve the paper. Derek Nowrouzezahrai helped with the final writing. We also thank the infographists, Laura Paiardini and Estelle Charleroy, for their help to design our prototype application. This work was partially funded by the advanced grant EXPRESSIVE from the European Research Council (ERC-2011-ADG_20110209), an Exploradoc grant from Région Rhône-Alpes, and the GRAND NCE in Canada.

References

[Bei06] BEISEL JR R. H.: *International Waterfall Classification System*. Outskirts Press, 2006. 4

[BMV*11] BERNHARDT A., MAXIMO A., VELHO L., HNAIDI H., CANI M.-P.: Real-time terrain modeling using CPU-GPU coupled computation. In *Proc. on Graphics, Patterns and Images (SIBGRAPI)* (2011), IEEE, pp. 64–71. 3

[BN08] BRUNETON E., NEYRET F.: Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum (Eurographics) 27*, 2 (2008), 311–320. 3, 10

[BSHK04] BHAT K., SEITZ S., HODGINS J., KHOSLA P.: Flow-based video synthesis and editing. *ACM Trans. on Graphics (SIGGRAPH) 23*, 3 (2004), 360–363. 2

[BSW10] BAGAR F., SCHERZER D., WIMMER M.: A layered particle-based fluid model for real-time rendering of water. *Computer Graphics Forum (EGSR) 29*, 4 (2010), 1383–1389. 2

[BTHB06] BENEŠ B., TĚŠÍNSKÝ V., HORNYŠ J., BHATIA S. K.: Hydraulic erosion. *Computer Animation and Virtual Worlds 17*, 2 (2006), 99–108. 3

[CEW*08] CHEN G., ESCH G., WONKA P., MÜLLER P., ZHANG E.: Interactive procedural street modeling. *ACM Trans. on Graphics 27*, 3 (2008), 103. 2

[CM10] CHENTANEZ N., MÜLLER M.: Real-time simulation of large bodies of water with small scale details. In *Proc. Symposium on Computer Animation* (2010), pp. 197–206. 2

[DD06] DANIELSSON M., DANIELSSON K.: *Waterfall Lover's Guide Northern California*. Mountaineers Books, 2006. 4

[EBP*12] EMILIEN A., BERNHARDT A., PEYTAVIE A., CANI M.-P., GALIN E.: Procedural generation of villages on arbitrary terrains. *The Visual Computer 28*, 6-8 (2012), 809–818. 2

[GCZ*06] GUAN Y., CHEN W., ZOU L., ZHANG L., PENG Q.: Modeling and rendering of realistic waterfall scenes with dynamic texture sprites. *Computer Animation and Virtual Worlds 17*, 5 (2006), 573–583. 2

[GGG*13] GÉNEVAUX J.-D., GALIN E., GUÉRIN E., PEYTAVIE A., BENEŠ B.: Terrain generation using procedural models based on hydrology. *ACM Trans. on Graphics (SIGGRAPH) 32*, 4 (2013), 143:1–13. 2, 3, 9, 10

[GM01] GAMITO M. N., MUSGRAVE F. K.: Procedural landscapes with overhangs. In *10th Portuguese Computer Graphics Meeting* (2001), vol. 2. 3, 10, 14

[GMS09] GAIN J., MARAIS P., STRASSER W.: Terrain sketching. In *Proc. Symposium on Interactive 3D Graphics and Games* (2009), ACM, pp. 31–38. 3

[GPGB11] GALIN E., PEYTAVIE A., GUÉRIN E., BENEŠ B.: Authoring hierarchical road networks. *Computer Graphics Forum (Pacific Graphics) 30*, 7 (2011), 2021–2030. 2

[GPMG10] GALIN E., PEYTAVIE A., MARÉCHAL N., GUÉRIN E.: Procedural generation of roads. *Computer Graphics Forum 29*, 2 (2010), 429–438. 14

[HGA*10] HNAIDI H., GUÉRIN E., AKKOUCHE S., PEYTAVIE A., GALIN E.: Feature based terrain generation using diffusion equation. *Computer Graphics Forum (Pacific Graphics) 29*, 7 (2010), 2179–2186. 3, 9

[HW04] HOLMBERG N., WÜNSCHE B. C.: Efficient modeling and rendering of turbulent water over natural terrain. In *Proc. Intl. Conf. on Computer Graphics and Interactive Techniques in Australasia and South East Asia* (2004), ACM, pp. 15–22. 2, 3

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. *ACM SIGGRAPH Computer Graphics 24*, 4 (1990), 49–57. 2

[KW06] KIPFER P., WESTERMANN R.: Realistic and interactive simulation of rivers. In *Proc. Graphics Interface* (2006), Canadian Information Processing Society, pp. 41–48. 2

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. on Graphics (SIGGRAPH) 23*, 3 (2004), 769–776. 3

[LH10] LEE H., HAN S.: Solving the shallow water equations using 2D SPH particles for interactive applications. *The Visual Computer 26*, 6-8 (2010), 865–872. 2

[LRBP12] LONGAY S., RUNIONS A., BOUDON F., PRUSINKIEWICZ P.: TreeSketch: Interactive procedural modeling of trees on a tablet. In *Proc. Symposium on Sketch-Based Interfaces and Modeling* (2012), pp. 107–120. 2

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. *ACM Trans. on Graphics 27*, 3 (2008), 102:1–10. 2

[MDH07] MEI X., DECAUDIN P., HU B.-G.: Fast hydraulic erosion simulation and visualization on gpu. In *Proc. Pacific Graphics* (2007), IEEE, pp. 47–56. 3

[PGGM09] PEYTAVIE A., GALIN E., GROSJEAN J., MERILLOU S.: Arches: a framework for modeling complex terrains. *Computer Graphics Forum (Eurographics) 28*, 2 (2009), 457–467. 3, 14

[Plu05] PLUMB G. A.: *Waterfall Lover's Guide Pacific Northwest*. Mountaineers Books, 2005. 4

[ŠBBK08] ŠŤAVA O., BENEŠ B., BRISBIN M., KŘIVÁNEK J.: Interactive terrain modeling using hydraulic erosion. In *Proc. Symposium on Computer Animation* (2008), pp. 201–210. 3, 14

[SDKT*09] SMELIK R., DE KRAKER K., TUTENEL T., BIDARRA R., GROENEWEGEN S.: A survey of procedural methods for terrain modelling. In *Proc. CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)* (2009). 3

[SDZ*07] SAKAGUCHI R., DUFOR T., ZALZALA J., LAMBERT P., KAPLER A.: End of the world waterfall setup for "Pirates of the Caribbean 3". In *ACM SIGGRAPH Sketches* (2007), ACM. 2

[Sta99] STAM J.: Stable fluids. In *Proc. SIGGRAPH* (1999), pp. 121–128. 11

[STBB14] SMELIK R. M., TUTENEL T., BIDARRA R., BENES B.: A survey on procedural modelling for virtual worlds. *Computer Graphics Forum* (2014). 2

[TGM13] TASSE F. P., GAIN J., MARAIS P.: Enhanced texture-based terrain synthesis on graphics hardware. *Computer Graphics Forum 31*, 6 (2013), 1959–1972. 3

[TMFSG07] THUREY N., MÜLLER-FISCHER M., SCHIRM S., GROSS M.: Real-time breakingwaves for shallow water simulations. In *Proc. Pacific Graphics* (2007), IEEE, pp. 39–46. 2

[vH11] VAN HOESEL F.: Tiled directional flow. In *ACM SIGGRAPH Posters* (2011), p. 19:1. 3, 11

[YNBH09] YU Q., NEYRET F., BRUNETON E., HOLZSCHUCH N.: Scalable real-time animation of rivers. *Computer Graphics Forum 28*, 2 (2009), 239–248. 2, 3, 14

[YNS11] YU Q., NEYRET F., STEED A.: Feature-based vector simulation of water waves. *Computer Animation and Virtual Worlds 22*, 2-3 (2011), 91–98. 3

[ZIH*11] ZHU B., IWATA M., HARAGUCHI R., ASHIHARA T., UMETANI N., IGARASHI T., NAKAZAWA K.: Sketch-based dynamic illustration of fluid systems. *ACM Trans. on Graphics (SIGGRAPH Asia) 30*, 6 (2011), 134:1–8. 3, 6, 7

[ZSTR07] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. *IEEE Trans. Visualization and Computer Graphics 13*, 4 (2007), 834–848. 3